A Tool for Hazard Detection in Hybrid Systems[*]

Peter Herrmann[†]
Universität Dortmund
Fachbereich Informatik, LS4
Peter.Herrmann@cs.uni-dortmund.de

Peter Grannas
MATERNA Information & Communications
Dortmund
Peter.Grannas@materna.de

Abstract

The complexity degree of modern chemical plants demands for the use of formal specification methods. A framework for hybrid systems contains specification modules and verification elements proving the plant safety. The design of a plant is reduced to the composition of framework components, the identification of possible sources of danger and the identification of suitable verification elements.
Our contribution introduces a tool supporting the selection of suitable safety properties eliminating possible sources of danger. The tool **harmonic** supporting this process is based on approaches of expert systems. An example examination of a plant specification clarifies the use of this program.

Keywords: Hazard Detection, Formal Methods, Specification Framework, Expert System, Harmonic.

1 Introduction

Since the operation of many hybrid chemical systems entails diverse risks, the design of plants has to be accompanied with careful and responsible hazard analysis procedures. Today, hazard analysis mostly follows the established method of "hazard and operability studies" (HazOps) [23]. In this informal approach teams of experts examine systematically descriptions of hybrid systems in order to detect subtle design faults which may cause hazards. Moreover, the teams try to detect the sources of the faults, predict consequences, and develop counter-measures. In order to facilitate HazOp studies, tool-assistance is proposed based on expert systems [5, 8, 26, 29] and on simulation [6, 24, 31]. Other approaches support the model-based formal analysis in order to achieve formal safety proofs of hybrid technical systems [3, 25, 27, 31]. Corresponding tool-support is based on exhaustive state space exploration [28] and on symbolic model checking [2, 22, 25]. The tools, however, are not satisfactory for the analysis of complex real-life systems since automated state space exploration fails due to the high number of reachable system states.

Our approach aims at the efficient formal verification of complex hybrid systems. It is based on temporal logic specifications [11] and on symbolic logical theorem proving [10]. Formal system specification and verification is facilitated by re-using specification modules (generic process type definitions) and verification elements (theorems) [13]. Adopting the notion of "frameworks" from software engineering (cf. [17]) and transferring it to the construction of formal models and proofs, we developed a so-called "hazard analysis specification framework" [15]. Besides of architectural rules this framework comprises three collections of specification resp. verification elements supporting the development of hybrid system specifications and proof scripts. One collection contains specification modules describing components used by chemical plants (e.g., vessels, valves, pumps, sensors, controllers). Plant specifications are developed quite easily by instantiation and composition of specification modules from this collection. The second collection consists of modules modeling safety properties to be fulfilled by a plant (e.g., a heating in a vessel may not run if the amount of liquid in the vessel is beyond a minimum value in order to prevent damage). The third collection contains generic theorems each guaranteeing that a safety property is provided by a plant subsystem modeled by a composition of component specification modules. The hazard analysis expert can reduce a proof of safety properties into lemmas corresponding directly to framework theorems. Since we already proved the validity of the theorems, he only has to perform some simple checks guaranteeing the consistency of the theorem and the plant model. Thus, the framework approach does not only facilitate the construction of formal plant specifications but also formal reasoning. For example, we specified an ethyl acetate production plant (cf. [6]) and proved 33 safety properties within four days.

The application of the framework in practice made it clear that an expert can further be supported by a rule-based tool facilitating the selection of suitable safety property modules to be proven. In this contribution we introduce the tool **harmonic** [7] which examines plant specifications composed from framework specification modules for possible system hazards. For instance, any system structure containing a pump linked to a valve is hypothetically a hazard source since a liquid may be pumped against the closed valve causing excess pres-

[*]This work was funded by the German research foundation DFG.
[†]Author to whom correspondence should be addressed.

sure. If **harmonic** detects a hazard source, it suggests a corresponding safety module from the framework (e.g., a liquid may not be pumped against a closed valve). If this safety property is fulfilled by the plant (e.g., if the system contains a controller switching off the pump if the valve is closed), the expert can perform a safety proof by means of a framework theorem. Otherwise, he has to adapt the plant in order to prevent this kind of hazard. **harmonic** examined the ethyl acetate plant on a Sun UltraSPARC-II (300 MHz) within 7 seconds suggesting 45 safety properties for verification.

The framework and **harmonic** are based on the temporal logic specification language cTLA [11, 14] which was extended from Leslie Lamport's "Temporal Logic of Actions" (TLA) [21]. cTLA uses a compositional process concept supporting the modular description of processes both in a resource-oriented and a constraint-oriented specification style [30]. Process composition has the character of superposition [4, 18]. Here, relevant properties of processes and subsystems are also relevant properties of the embedding system. Thus, superposition enables the use of framework theorems for safety proofs since a safety property fulfilled by a certain plant subsystem is also guaranteed by any entire plant specification containing the plant subsystem.

In the remainder we sketch the hazard analysis framework and cTLA. Afterwards, we introduce **harmonic** and outline an example application.

2 Framework for Hybrid Systems

Specification modules and theorems comprised by the hazard analysis framework [13, 15] (cf. WWW: `ls4-www.cs.uni-dortmund.de/RVS/P-HYSYS`) facilitate the formal specification and verification of hybrid technical systems. As depicted in Fig. 1 the set of specification modules consists of two collections for plant models resp. safety property modules both modeled by cTLA processes. A specification of a plant system is composed from plant model instantiations. A group of plant model modules is devoted to continuous, discrete, and hybrid system components which are mainly specified in a constraint-oriented way (cf. [30]). For instance, a vessel can be modeled by three separate plant model instances describing the volume of a liquid in the vessel, the temperature of this liquid, and the pressure in the vessel. Another group of plant modules specifies component malfunctions to be tolerated by a hybrid system (e.g., leakage of valves, jam of pipes, failures of active components like pumps or heatings).

The safety property modules are used to develop safe plant specifications describing a list of safety properties to be fulfilled by a hybrid system in order to prevent serious system hazards. A module states either that a system component observes a critical limit (e.g., the power of a vessel heating does not exceed 6000 *Watt* to prevent overheat) or that a critical system state will be excluded (e.g., a pump is switched off if the flow of

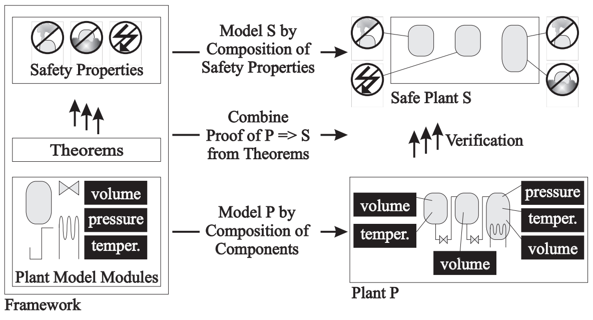

Figure 1: The framework for the formal hazard analysis of chemical plants

pumped liquid is blocked due to a closed valve). A formal description of a single safety property is developed by instantiating a safety property module, while a complete safe plant specification is derived by composing these module instances.

The formal proof that a plant system specification fulfills the safety properties in a safe plant specification is performed by means of framework theorems which are already proven implications of the form "plant subsystem $\wedge$ malfunction subsystem $\wedge$ safety property subsystem $\Rightarrow$ safety property" asserting that a certain plant subsystem guarantees a safety property even if certain malfunctions occur. For instance, the scenario that a liquid is pumped against a closed valve, is reflected by a framework theorem. In this theorem, the plant subsystem consists of the pump, the sensor, the controller, the pipe of pumped liquid, and appropriate links between these components. Since this control subsystem may tolerate a pipe jam but not controller or sensor failures, the malfunction subsystem contains only a plant model describing pipe jams. The safety property subsystem was introduced since plant components often interfere with many other and, perhaps, very distant components causing lengthy plant subsystem descriptions. In order to prevent this and, consequently, a large number of framework theorems, some theorems contain an additional conjunct of safety property modules at the premise describing safety requirements to the subsystem environment. These safety properties are verified transitively by using other framework theorems. For instance, consider a proof that a number of serially linked vessels never run empty. Without using a safety property subsystem in theorems, we would need a separate theorem for each line of vessels. The proof, however, can be performed easily by inductive application of a theorem stating that a vessel may not run empty under the safety requirement that its predecessor does not become empty as well.

3 cTLA

The safety property modules as well as the plant modules of the framework are specified by cTLA process type specifications [14] which model system behaviors by means of state transition systems. A cTLA process type contains private state variables describing the system state and actions which are predicates on a pair of a current and a next state and specify a set of transitions. Some additional cTLA constructs [11] which correspond to similar TLA concepts [1, 20] are used to express liveness, real time, and continuous system behavior. Liveness is specified by fairness assumptions on actions stating that an action may not be enabled infinitely often without being eventually executed. Similarly, a real time construct states that an action must be executed if it is enabled for a certain period of time. A special action is used to model continuous flows by means of difference equations. Furthermore, a process type may contain a list of generic process parameters in order to specify a spectrum of similar process instances.

For clarification Fig. 2 sketches two cTLA process types of the hazard analysis framework. The process type *ValveFlow* models a simple binary valve which allows the flow of a liquid only if it is open. It contains two generic process parameters. The parameter `initvalve` describes if the valve is open or closed in its initial state. The maximum flow of liquid through the valve is determined by `maxflow`. The current state of the valve is modeled by a variable `valve` which is defined in the part `VARIABLES`. The special predicate `INIT` specifies the set of initial variable states. `valve` corresponds initially to the generic parameter `initvalve`.

```
Binary valve within a line between two vessels
PROCESS ValveFlow
          ( initialvalve : {"closed","open"} ;
            initial state of valve
            maxflow : real )
            maximum flow through valve
BODY
  VARIABLES
    valve : {"closed","open"}; current state of valve
  INIT ≜ valve = initialvalve;
  ACTIONS
    CONT (OUTPUT flow : real) ≜ flow through line
      flow = IF (valve = "closed")
              THEN 0 ELSE maxflow ∧
      valve′ = valve;
    close ≜ close valve
      valve′ = "closed";
    open ≜ open valve
      valve′ = "open";
END

Maximum time for a controller to switch an actor in [sec]
PROCESS ReactMaxTime ( maxtime : real ) maximum
time
BODY
  ACTIONS
    switch;
  V MAX TIME switch : maxtime;
END
```

Figure 2: cTLA process types *ValveFlow* and *ReactMaxTime*

```
PROCESS CloseValveTimely

PROCESSES
  V  : ValveFlow ("closed",0.01);
  CT : ReactMaxTime (2);

ACTIONS
  CONT (OUTPUT flow) ≜
    V.CONT (; flow) ∧ CT.stutter;
  close ≜ V.close ∧ CT.switch;
  open ≜ V.open ∧ CT.stutter;
END
```

Figure 3: cTLA system type *CloseValveTimely*

The process contains the three actions `CONT`, `close`, and `open` which are listed in the part `ACTIONS`. Here, variables modeling the current state are described by the variable identifier (fi. `valve`) while variables specifying the next state are supplemented by the special symbol $'$ (fi. `valve′`). The action `CONT` describes the continuous behavior of the valve. It contains a special action parameter `flow` modeling the current flow through the valve which corresponds to 0 if the valve is closed and to the maximum flow `maxflow` if it is open[1]. The actions `close` and `open` model the closing resp. opening of the valve. The process type *ReactMaxTime* models a real time constraint. The construct `V MAX TIME` states that the action `switch` has to be executed before it is continuously enabled for `maxtime` time units.

cTLA processes are composed to (sub)system descriptions (fi. plant system and safe plant system specifications in the hazard analysis framework). As in the formal description technique LOTOS, interactions between these processes are modeled by joint system actions which are coupled from local process actions. The process actions coupled to the same system action have to be executed simultaneously. Data transfer between processes in a system is described by action parameters. Fig. 3 describes a simple subsystem specification which is modeled by the cTLA system specification type *CloseValveTimely*. It describes that a valve may not be open for longer than 2 time units. The subsystem is composed of two process instances listed in the part `PROCESSES`. The valve is modeled by the process instance *V* which is instantiated from the cTLA process *ValveFlow* in Fig. 2. The generic process parameters `initvalve` and `maxflow` are replaced by the values `"closed"` resp. 0.01. The process *CT* of type *ReactMaxTime* models the timing constraint guaranteeing the timely closing of the valve. The coupling of local process actions to joint system actions is specified in the part `ACTIONS`. The system contains the system actions `CONT`, `close`, and `open` modeling the continuous behavior and the closing resp. opening of the valve. Since *V* is the only instance modeling continuous flows, `CONT` consists of the local action `CONT` from process *V*

[1]The flow through an open valve may fall below the maximum value. This, however, is specified by separate cTLA processes.

while *CT* performs a so-called stuttering step, in which its local variables are not changed. The interaction between the two processes is modeled by the system action `close`. By incorporating the action `close` of process instance *V* one guarantees that this system action models the closing of the valve. The action `switch` of process *CT* assures that the system action is executed within 2 time units. The system action `open` consists of the process action `open` of *V* while *CT* performs a stuttering step.

4 HARMONIC

The identification of suitable safety properties is supported by the tool **harmonic** (hazard recognition tool to model technical systems based on cTLA) [7]. It follows the knowledge-based approach introduced in [19]. To solve the problem of hazard detection it consists of five knowledge-bases which can be easily changed and adapted to the framework components. Safety properties are determined from the given system specification. To identify suitable properties, typical process structures, like dependencies between different components (e.g. a valve and a pump in one line), have to be identified. This is been done by support of rules which are stored in the knowledge-base production rules.

To identify these structures, a flow graph is used. This graph is computed from the plant specification. Nodes are sources and sinks of liquid, gas, energy or pressure flows specified by cTLA-processes (e.g., the process `VesselVolume` modeling the volume of liquid in a vessel is a source and sink of liquid flows). Edges model flows and are specified by cTLA-processes like the flow depending on the valve (`ValveFlow`) and on the pump (`PumpFlow`). These processes define real flows existing in a chemical plant. Which components of the framework are nodes of the flow graph, which of their parameters define an incoming resp. outgoing flow and which components define flows is been told to the tool **harmonic** by support of the flow graph knowledge.

Rules use functions like `areInFlows`, `isOutFlow` or `inSys`. These functions are implemented in source code and examine the flow graph. The syntax of the functions is defined in the function knowledge-base and used for parsing purposes of the production rules. For instance, the rule

```
areInFlows(ValveFlow, PumpFlow, VesselVolume)
    :: ValveOpenorNoPump(#);
```

states if a valve (`ValveFlow`) and a pump (`PumpFlow`) are connected in line (`areInFlows`) to a vessel (`VesselVolume`), the safety property `ValveOpenOrNoPump(#)` has to be fulfilled by the system specification to prevent a pump running against a closed valve.

All knowledge-bases are easy adaptable to framework changes. So, good maintenance, changeability and flexibility are provided. Framework components

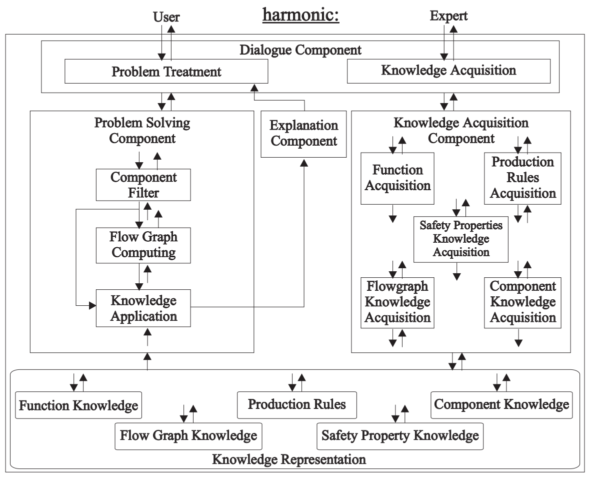

Figure 4: Structure model

defining plant constraints are stored in the base component knowledge, and framework safety properties are saved in the safety property knowledge-base. Elements of these two bases are specified in cTLA.

The acquisition of all five knowledge-bases is been done by the dialogue component (cf. Fig. 4), which implements an interface to a knowledge expert, and the knowledge acquisition component, which prepares the information for internal computing. This information is stored in the knowledge representation to produce them to the problem solving component. The latter component gets the plant system specification from a user through the dialogue component. The specification is prepared for internal computations (component filter) and the flow graph is calculated (flow graph computing). After these steps the production rules are applied and the results are forwarded to an explanation component for giving possible explanations of the results. Finally the results are passed back to the user through the interface dialogue component.

5 EXAMPLE

In this section we give a small impression of the functionality of **harmonic**. Fig. 5 depicts a part of an ethyl acetate plant (cf. [6]). The volume of vessel `F1` is modeled by a cTLA process `F1FVol` (of type `VesselVolume`) and the volume of `D1` by `D1FVol`

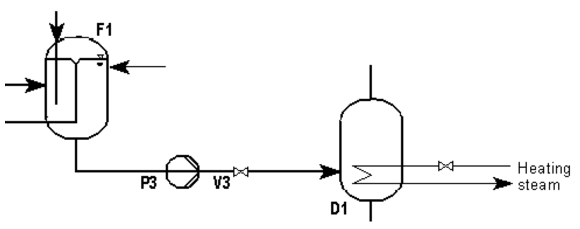

Figure 5: PID of a plant part

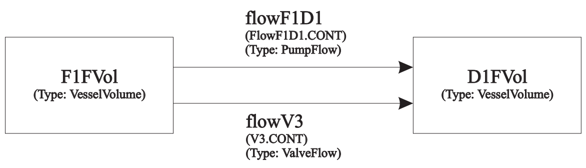

Figure 6: Flow graph

(`VesselVolume`). The pump `P3` is specified by the framework component `FlowF1D1` (of type `PumpFlow`) and the valve `V3` by the process `V3` (`ValveFlow`). Application of the flow graph knowledge identifies the processes `FlowF1D1` (of type `PumpFlow`) and `V3` (`ValveFlow`) for flow definition. They define the flow names `flowF1D1` resp. `flowV3`. In this example, sources and sinks are specified by the process type `VesselVolume` and flows are defined by process types `ValveFlow` and `PumpFlow`. Fig. 6 depicts the corresponding flow graph part. Rules defined in the production rule base can be applied on this graph after its computation:

```
inSys(ValveFlow) :: LineMaxFlow(1.0);
isInFlow(ValveFlow, VesselVolume)
    :: VesselMaxPressureOrNoFlow(#);
areInFlows(ValveFlow, PumpFlow, VesselVolume)
    :: ValveOpenorNoPump(#);
```

The results are shown in Fig. 7. The first rule checks whether processes of type `ValveFlow` are defined anywhere in the specification. It contains the safety property `LineMaxFlow` with parameter `1.0`. This safety property assures that the flow through the valve `V3` does not exceed a limit of 1.0 m^3/sec. The second rule states that for all flows which are of type `ValveFlow`

```
Rule causing the following output:
inSys(ValveFlow) :: LineMaxFlow(1.0)
Component:  File:  Safety-Property:          Params:
-----------------------------------------------------
V3          PS.ct  LineMaxFlow               (1.0)

Rule causing the following output:
isInFlow(ValveFlow, VesselVolume)
  :: VesselMaxPressureOrNoFlow(#)
Component:  File:  Safety-Property:          Params:
-----------------------------------------------------
V3          PS.ct  VesselMaxPressureOrNoFlow  (#)
D1FVol      PS.ct  VesselMaxPressureOrNoFlow  (#)

Rule causing the following output:
areInFlows(ValveFlow, PumpFlow, VesselVolume)
  :: ValveOpenOrNoPump(#)
Component:  File:  Safety-Property:          Params:
-----------------------------------------------------
V3          PS.ct  ValveOpenOrNoPump          (#)
FlowF1D1    PS.ct  ValveOpenOrNoPump          (#)
D1FVol      PS.ct  ValveOpenOrNoPump          (#)
```

Figure 7: Results

and incoming flows of processes of type `VesselVolume` the safety property `VesselMaxPressureOrNoFlow` has to be fulfilled. This property assures that a flow through a valve connected to a vessel occurs only if the pressure in the vessel does not exceed a certain value. Since the safety property parameter is not defined (#), the plant developer adjusts the parameter after the plant evaluation. Due to the second rule **harmonic** suggested processes `V3` and `D1FVol` to the system developer for checking with safety property `VesselMaxPressureOrNoFlow`. The last rule gets all processes of type `ValveFlow` and `PumpFlow` which are incoming flows of processes of type `VesselVolume`. Therefore it is possible to check whether a valve and a pump are connected in one line to a vessel. The property `ValveOpenOrNoPump` assures that a pump does only work for a certain period of time if the flow in a line is blocked due to a closed valve. In our example valve `V3` (process: `V3`, flow name: `flowV3`) and pump `P3` (`FlowF1D1`, `flowF1D1`) are connected to vessel `D1` (`D1FVol`).

The safety properties are computed for each process found during the evaluation of one rule. Since components can be specified in different subsystems, the expert has to decide for which subsystem or subsystems he proves the detected safety property (e.g., in the third rule valve `V3` and pump `P3` (`FlowF1D1`) could be specified in `Subsystem1` and vessel `D1` (`D1FVol`) in `Subsystem2`).

6 CONCLUDING REMARKS

The approach of specification frameworks, which was also used in the computer science field of telecommunication protocols [12, 14], proved as a successful means in supporting formal hazard analysis of hybrid technical systems. **harmonic** facilitates the framework application by suggesting useful safety properties to be kept by a plant in order to avoid hazards. It complements two other tools. The cTLA-program ctc [16] enables the composition of single specification modules to complete subsystem or system descriptions. The tool COAST [9] supports formal proofs by the selection of suitable framework theorems and carrying out the theorem consistency checks.

REFERENCES

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543-1571, 1994.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic Symbolic Verification of Embedded Systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.

[3] C. A. Catino and L. H. Ungar. A model-based approach to automated hazard identification of chemical plants. *AIChE Journal*, 41(3):97–109, 1995.

[4] K. M. Chandy and J. Misra. *Parallel Program Design — A Foundation*. Addison Wesley, 1988.

[5] M. Göring and H. G. Schecker. HAZEXPERT: An integrated expert system to support hazard analysis in process plant design. *Computers Chemical Engineering*, 17:429–434, 1993.

[6] H. Graf and H. Schmidt-Traub. A Model-Based Approach to Process Hazard Identification. In *Proc. 13th International Congress of Chemical and Process Engineering*, Prague, 1998.

[7] P. Grannas. Ein rechnergestütztes System zur Erkennung von Hazards in hybriden technischen Systemen. Diploma Thesis, Universität Dortmund, FB Informatik, 2000 (in German).

[8] P. Heino, A. Poucet, and J. Soukas. Computer tools for hazard identification, modelling and analysis. *Journal of Hazardous Materials*, 29:445–463, 1992.

[9] P. Herrmann, O. Drögehorn, W. Geisselhardt, and H. Krumm. Tool-supported formal verification of high-speed transfer protocol designs. In *Proc. 7th International Conference on Telecommunication Systems — Modelling and Analysis*, pages 531–541, Nashville, 1999. ATSMA.

[10] P. Herrmann, G. Graw, and H. Krumm. Compositional Specification and Structured Verification of Hybrid Systems in cTLA. In *Proc. 1st IEEE International Symposium on Object-oriented Real-time distributed Computing*, pages 335–340, Kyoto, 1998. IEEE Computer Society Press.

[11] P. Herrmann and H. Krumm. Specification of Hybrid Systems in cTLA+. In *Proc. 5th International Workshop on Parallel & Distributed Real-Time Systems*, pages 212–216, Geneva, 1997. IEEE Computer Society Press.

[12] P. Herrmann and H. Krumm. Modular Specification and Verification of XTP. *Telecommunication Systems*, 9(2):207–221, 1998.

[13] P. Herrmann and H. Krumm. Formal Hazard Analysis of Hybrid Systems in cTLA. In *Proc. 18th IEEE Symposium on Reliable Distributed Systems*, pages 68–77, Lausanne, 1999. IEEE Computer Society Press.

[14] P. Herrmann and H. Krumm. A Framework for Modeling Transfer Protocols. To appear in *Computer Networks*, 2000.

[15] P. Herrmann and H. Krumm. A Framework for the Hazard Analysis of Chemical Plants. To appear in *Proc. 11th IEEE International Symposium on Computer-Aided Control System Design (CACSD2000)*, Anchorage, Omnipress, 2000. IEEE CSS.

[16] C. Heyl, A. Mester, and H. Krumm. ctc — A Tool Supporting the Construction of cTLA-Specifications. In T. Margaria and B. Steffen, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1055, pages 407–411. Springer-Verlag, 1996.

[17] R. E. Johnson and B. Foote. Designing reusable classes. *The Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[18] R. Kurki-Suonio. Hybrid Models with Fairness and Distributed Clocks. In R. L. Grossmann, A. Nerode, A. Ravn, and H. Rischel, editors, *Proc. Workshop on Theory of Hybrid Systems*, LNCS 736, pages 103–120, Lyngby, 1993. Springer-Verlag.

[19] R. Lackes. Simulation and knowledge-based analysis of Just-in-Time-production systems. In *Proc. 20th International Conference on Computers & Industrial Engineering*, Seoul, 1996.

[20] L. Lamport. Hybrid Systems in TLA$^+$, In R. L. Grossmann, A. Nerode, A. Ravn, and H. Rischel, editors, *Proc. Workshop on Theory of Hybrid Systems*, LNCS 736, pages 77–102, Lyngby, 1993. Springer-Verlag

[21] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[22] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1((1+2)), 1997.

[23] H. G. Lawley. Operability Studies and Hazard Analysis. *Chemical Engineering Progress*, 70(4):45–56, 1974.

[24] N. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, 1995.

[25] S. T. Probst. *Chemical Process Safety and Operability Analysis using Symbolic Model Checking*. PhD thesis, Carnegie Mellon University, Pittsburgh, 1996.

[26] Y. Shimada, K. Suzuki, and H. Sayama. Computer-aided operability study. *Computers Chemical Engineering*, 20(6/7):905–913, 1996.

[27] R. Srinivasan and V. Venkatasubramanian. Petri Net-Digraph models for automating HAZOP analysis of batch process plants. *Computers Chemical Engineering*, 20:719–725, 1996.

[28] O. Stursberg, H. Graf, S. Engell, and H. Schmidt-Traub. A concept for safety analyses of chemical plants based on discrete models with an adapted degree of abstraction. In *Proc. 4th International Workshop on Discrete Event Systems*, Cagliari, 1998.

[29] R. Vaidhyanathan and V. Venkatasubramanian. Experience with an expert system for automated HAZOP analysis. *Computers Chemical Engineering*, 20:1589–1594, 1996.

[30] C. A. Vissers, G. Scollo, and M. van Sinderen. Architecture and specification style in formal descriptions of distributed systems. In S. Agarwal and K. Sabnani, editors, *Proc. Protocol Specification, Testing and Verification VIII*, pages 189–204, Elsevier, 1988. IFIP.

[31] A. Waters and J. W. Ponton. Qualitative simulation and fault propagation in process plants. *Chemical Engineering Research Descriptions*, 67:407–422, 1989.